ERDC/CRREL TR-06-20

US Army Corps
of Engineers®
Engineer Research and
Development Center

Object-Oriented Approach to Manipulating Acoustic and Seismic Spectra

D. Keith Wilson and Jacob I. Torrey

December 2006

Cold Regions Research
and Engineering Laboratory

ERDC/CRREL TR-06-20
December 2006

Object-Oriented Approach to Manipulating Acoustic and Seismic Spectra

D. Keith Wilson and Jacob I. Torrey

Cold Regions Research and Engineering Laboratory
U.S. Army Engineer Research and Development Center
72 Lyme Road
Hanover, NH 03755-1290

Prepared for U.S. Army Corps of Engineers

Abstract: The software design and underlying mathematics for an object-oriented, Java-based approach to creating and manipulating frequency-dependent functions, such as power spectral densities, is described. The frequency dependence is modeled as a series of power-law bands, which provides a high degree of flexibility and efficiency for representing common spectral models such as evenly spaced bands, octave bands, narrow spectral lines, broadband noise, and power laws. Conversions between the various spectral models are easily performed. Many common operations on spectra, such as filtering, incoherent addition, application of transfer functions, and calculation of signal-to-noise ratios, can be conveniently applied. While this capability was developed to serve as a basis for future development of tactical decision aids and mission planning tools for battlefield seismics and acoustics, many other applications involving spectra are possible.

Contents

Preface vi

1 Introduction 1

2 Background and Definitions 4

3 Spectral Representation Scheme and Java Implementation 7

a. Banded Power-Law Spectrum 7
b. Representing Common Spectral Models 8
White-noise bands 8
Spectral lines 8
Power-law spectra 9
Proportional bands 9
Discrete Fourier transform (DFT) 9
Broadband white noise 10
Broadband pink noise 10
Transfer functions 10
Dual power-law spectrum 11
c. Java Implementation 12
d. Constructors 14
e. Representing Frequency Bounds 14
f. Lower-Level Get/Set Methods 15
g. Clones, Equality Tests, and String Conversions 16
h. Evaluating the Spectrum 17

4 Signal Power and Moment Methods 18

a. Higher-Level Set Methods 18
b. Higher-Level Get Methods 21
c. Decibel Conversions 24

5 Filtering and Frequency Bands 25

a. Removing Bands and Eliminating Overlap 25
b. Low-, High-, Bandpass, and Stopband Filtering 25
c. Converting Frequency Bands 27

6 Multi-Spectra Operations 29

a. Developing Common Frequency Bounds 29
b. Adding Spectra Incoherently 31
c. Dividing Spectra 32
d. Multiplying Spectra 32

7 Conclusion 33

References....34

Report Documentation Page....35

Figures

Figure 1. Procedure for the frequency-domain calculation of the signal-to-noise ratio received by a seismic or acoustic sensor 1

Figure 2. Example of the banded, power-law spectral representation 7

Figure 3. Illustration of approximating the dual power-law spectrum with a set of bands described by single power laws 12

Figure 4. Determination of the unique bounding frequencies from a pair of Spec objects 30

Preface

This report was prepared by D. Keith Wilson and Jacob I. Torrey, Signature Physics Branch, Cold Regions Research and Engineering Laboratory (CRREL), U.S. Army Engineer Research and Development Center (ERDC), Hanover, NH.

This development was funded in part by the ERDC Battlefield Terrain Reasoning and Awareness AT42 work package. The authors also thank David Marlin of the Army Research Laboratory and Sean Mackay and Marah McClelland of Atmospheric and Environmental Research, Inc., for their comments and programming advice.

The report was prepared under the general supervision of Dr. Justin Berman, Acting Chief, Civil and Infrastructure Engineering Branch; Dr. Lance Hansen, Deputy Director; and Dr. Robert Davis, Director, CRREL.

The Commander and Executive Director of ERDC is COL Richard B. Jenkins. The Director is Dr. James R. Houston.

1 Introduction

Because of their low cost, low power consumption, wide field of regard, and other factors, acoustic and seismic sensors are becoming increasingly common for battlefield applications. However, their performance depends strongly on environmental transmission effects and background noise, both of which depend on the acoustic or seismic frequency. The energy emitted by acoustic and seismic sources also usually has a strong frequency dependence. It is thus typical to model the performance of the sensors with a frequency-domain calculation, in which the source emission, transmission, and noise are calculated at a number of frequencies independently.

The general flow of a frequency-domain, sensor performance calculation is illustrated in Figure 1. Models or field data are used to estimate the frequency spectra of the source (target) and the environmental background noise. The source spectrum is multiplied by a transfer function representing the environmental transmission effect, and then by a transfer function for the sensor response, to obtain the source signal spectrum as it appears to the sensor. The environmental background noise is multiplied by the transfer function for the sensor response and then added to the sensor self-noise spectrum to obtain the total noise spectrum at the sensor. Lastly, the apparent signal spectrum is divided by the noise spectrum to obtain a frequency-dependent signal-to-noise ratio, from which metrics of interest such as the probabilities of detection and false alarm can be calculated.

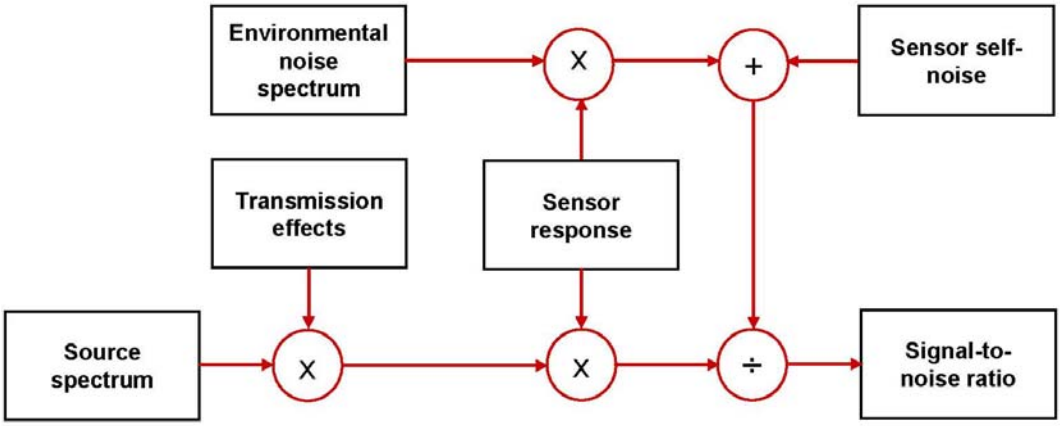

Figure 1. Procedure for the frequency-domain calculation of the signal-to-noise ratio received by a seismic or acoustic sensor.

The operations involved in Figure 1 include representing spectra (such as the source and environmental noise), multiplying by frequency-dependent transfer functions, adding spectra together, and dividing spectra. Although these operations are not all that complicated in principle, in practice they can be challenging to implement in a consistent and computationally efficient manner, due to the different methods for representing frequency bands in spectra and transfer functions. An example of a software tool that performs these operations is the Sensor Performance Evaluator for Battlefield Environments (SPEBE) (Wilson et al. 2002, Wilson 2006), which models the performance of battlefield acoustic and seismic sensors. Typically, the source spectrum is described most conveniently as a set of harmonically related bands, whereas the noise background is described with a broadband or octave band representation. Due to their computational intensiveness, the transmission effects are calculated at a fixed set of frequencies that do not usually align with the source and noise spectra. The sensor response and self noise are described by their own frequency-dependent behaviors, which usually are broadband or have transitions between two or more characteristics. By the time the signal-to-noise ratio is calculated, many conversions and operations involving different frequency-domain representations have been performed.

This report describes the formulation of new object-oriented software tools that are designed to conveniently, consistently, and efficiently manipulate spectral information. The software is implemented in the Java language because of its support for object-oriented programming and because it is the main language for the Battlefield Terrain Reasoning and Awareness (BTRA) project, which includes acoustic/seismic sensor performance prediction as one of its goals. It is also hoped that the new Java code can serve as a basis for future versions of SPEBE and for developing an acoustic/seismic computational engine to support other Army modeling and simulation efforts. Although acoustics and seismics are the explicit focus of this report, the approach to manipulating spectra and frequency-dependent functions is general enough to be useful in many other applications.

The new tools differ from currently available spectral processing packages in that they have been designed to easily accommodate spectral data of all kinds, including broadband noise, power laws, narrow harmonic lines, data stored in evenly spaced frequency bins, and octave-band data. Rou-

tines (methods) provide conversion between the various representations and manipulations including addition, multiplication, and division of the frequency-dependent functions.

This report is not intended as a replacement for the customary Java-style software documentation, known as JavaDocs. The purpose is rather to describe the conceptual approach and mathematics underlying the new Java code. It is expected that programmers will still refer to the JavaDocs for detailed information on the Java methods and data fields.

The next section of this report, Section 2, provides some general background and definitions related to spectra. This background will be important in describing the operation of the code. Next, in Section 3, the basic framework for manipulating spectra (fields, constructors, etc.) is discussed. Section 4 describes higher-level methods that primarily serve the purpose of setting and retrieving the frequency-dependent signal power. Methods for filtering spectra and selectively removing bands are described in Section 5. Lastly, in Section 6, we describe methods that operate on multiple spectra, including addition, multiplication, and division.

2 Background and Definitions

A spectrum describes the frequency content of a time-varying signal. Let us designate the time-varying signal as $x(t)$, where t is time. Suppose that $x(t)$ is passed through a perfect bandpass filter that removes all but the signal variations between a lower frequency f and an upper frequency $f + \Delta f$, where Δf is the analysis *bandwidth* of the filter. We designate the power that remains in the signal as $E(f, f + \Delta f)$. The (single-sided) *autospectral density* for the signal power is then defined as

$$S(f) = \lim_{\Delta f \to 0} \frac{E(f, f + \Delta f)}{\Delta f},$$

which implies

$$E(f, f + \Delta f) = \int_f^{f+\Delta f} S(f')\,df'. \tag{1}$$

The total power in the signal is found by integrating over all possible frequencies:

$$E(0, \infty) = \int_0^{\infty} S(f')\,df'. \tag{2}$$

For the purposes of this report, *power* generically refers to a quantity proportional to the squared amplitude of the signal. In acoustics, the sensors usually measure the pressure of a sound wave, in which case the amplitude is in Pa and the so-called power in Pa^2. In seismics, a geophone produces the velocity of a ground vibration, which has units ms^{-1} and the so-called power has units m^2s^{-2}. In either case, multiplication by other constant factors converts the squared-amplitude units to true power.

We define the mth moment of the spectral density, between two frequencies f_ℓ and f_u, as

$$M^m(f_\ell, f_u) = \int_{f_\ell}^{f_u} f^m S(f)\,df. \tag{3}$$

Note from (1) that $M^0(f_\ell, f_u) = E(f_\ell, f_u)$.

In many cases the spectral density is well described by a *power-law* relationship.* Specifically, $S(f)$ is described by an equation

$$S(f) = Af^p, \tag{4}$$

where A is a spectral coefficient and p the power-law exponent. This representation is useful for describing wideband white noise ($p = 0$), pink noise ($p = -1$), red noise ($p = -2$), and many other physical processes. For example, a turbulence spectrum usually has a frequency range over which $p = -5/3$. Substituting (4) into (3) and integrating, we find for the moments

$$M^m(f_\ell, f_u) = \int_{f_\ell}^{f_u} Af^{p+m}\, df = \begin{cases} \frac{A}{p+m+1}\left(f_u^{p+m+1} - f_\ell^{p+m+1}\right), & p+m \neq -1 \\ A\log\left(f_u / f_\ell\right), & p+m = -1 \end{cases}. \tag{5}$$

It is very common, particularly in acoustics, to represent a spectrum in *proportional* bands. In proportional bands, the ratio of the upper frequency bound f_u to the lower frequency bound f_ℓ is constant. For *octave bands*, $f_u / f_\ell = 2$. For *1/3-octave bands*, $f_u / f_\ell = 2^{1/3}$. In general, $f_u / f_\ell = 2^{1/\alpha}$, for 1/α-octave bands. The geometric center frequency of the band is defined as $f_g = \sqrt{f_\ell f_u}$, from which it follows that $f_g = 2^{1/2\alpha} f_\ell = 2^{-1/2\alpha} f_u$. The bandwidth (also called the *interval*) of the 1/α-octave, $\Delta f = f_u - f_\ell$, is therefore $(2^{1/2\alpha} - 2^{-1/2\alpha}) f_g = (2^{1/\alpha} - 1) f_\ell = (1 - 2^{1/\alpha}) f_u$. In acoustics, a standard set of analysis bands has been established. The standard geometric center frequencies for audible-range, octave-band analyses are 16, 31.5, 125, 250, 500, 1000, 2000, 4000, and 8000 Hz.

It is also common practice in acoustics to specify signal loudness in decibels (dB). Such quantities are referred to as *levels*. The sound-pressure level is related to $E(f_\ell, f_u)$ according to

* The word power as used here in power-law refers to an exponent in a mathematical equation. This should not be confused with the usage in the previous paragraph, where power referred to power in a signal.

$$L(f_\ell, f_u) = 10 \log_{10} \frac{E(f_\ell, f_u)}{p_r^2} \quad (6)$$

where $\Delta f = f_u - f_\ell$ is an analysis bandwidth and p_r (20 μPa for outdoor acoustics) is a standard reference pressure. The analysis bandwidth may be a unit band (1 Hz), or it may be an octave band or fraction thereof.

So far the discussion has focused on autospectral densities. Autospectral densities are equivalent to the Fourier transform of the correlation function between a signal, $x(t)$, and the same signal advanced in time, $x(t + \tau)$ (e.g., Bendat and Piersol 1986). One can also define a cross-spectral density, $S_{xy}(f)$, which is the Fourier transform of the correlation function between a signal, $x(t)$, and a different signal advanced in time, $y(t + \tau)$. The cross-spectral density, in general, is a complex number since it includes information on the relative phases of the signals as well as power. With the extension to complex functions, most of the previous discussion can apply to cross-spectral densities as well.

3 Spectral Representation Scheme and Java Implementation

a. Banded Power-Law Spectrum

The original ABFA program (Wilson and Szeto 2000), a tactical decision aid for battlefield acoustics, represented the spectral densities as the sum of the spectral densities in a number of discrete, non-overlapping bands. Each band was described by a power-law relationship. Mathematically,

$$S(f) = \sum_{n=1}^{N} S_n(f), \tag{7}$$

where

$$S_n(f) = \begin{cases} A_n f^{p_n}, & f_{\ell,n} \le f < f_{u,n} \\ 0, & \text{elsewhere.} \end{cases} \tag{8}$$

In these equations, $f_{\ell,n}$ is the lower frequency bound, $f_{u,n}$ is the upper frequency bound, A_n is the spectral coefficient, and p_n is the power-law exponent. The subscript n is the index of the frequency band. To accommodate cross-spectral densities and other complex-valued, frequency-dependent functions, phase information is also included in the representation. A single phase angle φ_n is stored for each band, which is interpreted as being constant throughout the band.

When the power-law bands are plotted on logarithmic axes, they appear as linear segments. An illustration is shown in Figure 2.

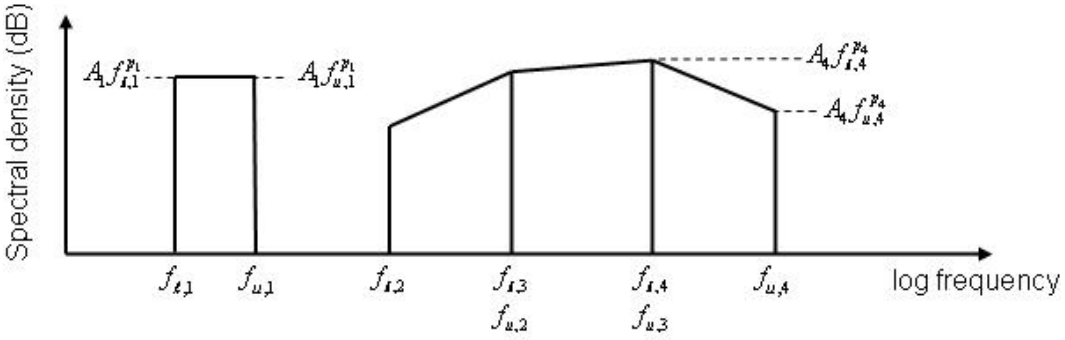

Figure 2. Example of the banded, power-law spectral representation. The frequency and power axes are shown logarithmically.

b. Representing Common Spectral Models

Many spectral models familiar from signal processing can be readily represented with the banded power-law scheme described in the previous section. Here we show how the variables ($f_{\ell,n}$, $f_{u,n}$, A_n, p_n, φ_n), $n = 1, \ldots, N$ can be set to obtain various common spectral models. A recurring theme is that it is often more convenient to specify the power in a band (relative to an analysis bandwidth of either $\Delta f_n = f_{u,n} - f_{\ell,n}$, a unit bandwidth, or a fractional octave band) than the spectral coefficients A_n. Therefore, it is desirable that the software implement conversions between power and the spectral coefficients. Such methods will be described later. For now, we assume that the conversions are available.

White-noise bands

In many situations we may regard the spectral density as a set of finite-width bands, where each band contains white (frequency-independent) noise. Mathematically, this situation is described by the equation

$$S(f) = \sum_{n=1}^{N} C_n \Pi\left(\frac{f_{u,n} - f_{\ell,n}}{\Delta f_n}\right) \tag{9}$$

where N is the number of bands, C_n is the spectral density of the particular band, and Π is 1 for arguments between 0 and 1 and is 0 otherwise. To store such a spectrum using the representation described in the previous section, the power-law exponents p_n are all set to zero. The A_n are equivalent to the C_n.

Spectral lines

This is a special situation where the power spectral density contains power at a set of infinitely narrow bands. Mathematically,

$$S(f) = \sum_{n=1}^{N} B_n \delta(f_n) \tag{10}$$

where B_n is the energy in the spectral line, $\delta()$ is the unit impulse, and f_n is the frequency of the line. To represent spectral lines, we could set the $f_{\ell,n} = f_n - \delta f/2$ and $f_{u,n} = f_n + \delta f/2$, where δf is very small. The signal power in each band, $E(f_{\ell,n}, f_{u,n}) = B_n$, is most conveniently specified (rather than the

spectral coefficients A_n), since it is independent of the somewhat arbitrary width of the bands. The power-law exponents are immaterial and therefore can all be set to zero.

Power-law spectra

This model is described directly by Eqs. (7) and (8), which extend (4) to the case of multiple power-law bands. The $f_{\ell,n}$ and $f_{u,n}$ are set according to the boundaries between the bands. The spectral coefficients A_n and slopes p_n are known directly.

Proportional bands

Presuming the signal power has been analyzed into the proportional bands, we would set the $f_{\ell,n}$ and $f_{u,n}$ to the corresponding lower and upper frequencies of the proportional bands. Typically, when an analysis into proportional bands is performed, there is no information on the variation of the spectrum within the bands and therefore p_n is unknown. There may be a preference to set the p_n to 0 to correspond to white noise in the bands, or to –1 for pink noise. The power in each band is normally specified, which can be converted to find the A_n.

Discrete Fourier transform (DFT)

Spectra are normally calculated on computers using DFTs. The DFT transforms a discretized, finite-length time signal to yield the complex Fourier coefficients

$$X_n,\ n = 1, 2, \ldots, N \tag{11}$$

where the X_n represent the Fourier spectrum at the discrete frequencies $f_n = n\Delta f$, $n = 0, \ldots, N-1$. The power spectral density is proportional to the square of the absolute values of the complex coefficients:

$$S(f_n) \propto \left| X(f_n) \right|^2 .$$

The constant of proportionality depends on conventions that are used to define the DFT. In this report, we assume that the DFT convention produces a *variance* spectrum such that

$$E(0, f_N) = \sum_{n=1}^{N} \left| X(f_n) \right|^2 .$$

Regarding the $X(f_n)$ as constant between f_n and $f_n + \Delta f$, we then have

$$S(f_n) = \left| X(f_n) \right|^2 / \Delta f . \tag{12}$$

In the case of DFT data, the signal has been transformed into bands of a fixed width Δf. The lower frequency bounds are $f_{\ell,n} = (n - 1)\Delta f$, $n = 1, \ldots, N$, and the upper frequency bounds are $f_{u,n} = n\Delta f$. Since no information is available on the variation of the spectral density within each band, we assume constant bands and set $p_n = 0$. This implies $A_n = \left| X(f_n) \right|^2 / \Delta f$. The phase angles are zero for autospectra. For cross spectra, they would be $\phi_n = \angle \left[X^*(f_{\ell,n}) Y(f_{\ell,n}) \right]$, where the asterisk indicates complex conjugation.

If desired, an ordinary DFT could also be stored directly. In this case, the A_n would simply be $X(f_n)$, and the phase angles $\phi_n = \angle X(f_{\ell,n})$. However, in this report we focus on spectral densities.

Broadband white noise

For broadband white noise (noise with a power spectral density that is independent of frequency over a wide frequency range), one would normally specify the signal power per unit bandwidth. The $f_{\ell,n}$ and $f_{u,n}$ are the lower and upper frequencies of the white noise band.

Broadband pink noise

For broadband pink noise (noise with a power spectral density proportional to f^{-1} over a wide frequency range), one would normally specify the signal power per octave or 1/3 octave. The p_n are set to –1. Note that the frequency range of the pink noise could be longer or shorter than an octave (or 1/3 octave), even though the power is specified for that analysis bandwidth.

Transfer functions

A transfer function refers to the ratio $H(f) = S_{xy}(f)/S_{xx}(f)$, where $S_{xx}(f)$ is the autospectral density of the input signal to a system $x(t)$ and $S_{xy}(f)$ is the cross-spectral density between the output $y(t)$ and the input. Hence a

transfer function is not a power spectral density, but rather just a frequency-dependent function. The transfer function can be partitioned into segments that are approximated by the functional form Eqs. (7) and (8) as desired.

Dual power-law spectrum

Typically, a power-law representation applies only to a range of frequencies. To represent a spectrum over a broad range of frequencies, a more complex representation must be used. For example, the following spectrum, originating with von Kármán (1948), is often used for turbulence:

$$S(f) = \frac{\sigma^2 \tau}{\left(1 + f^2 \tau^2\right)^{5/6}} \tag{13}$$

where σ^2 is the variance of the process and τ a time scale. This spectrum is an example of the general form

$$S(f) = \frac{Af^p}{\left(1 + B^2 f^2\right)^q} \tag{14}$$

in which A, B, p, and q are constants. For low frequencies, $Bf << 1$, this reduces to the power law $S(f) \simeq Af^p$. For high frequencies, $Bf >> 1$, it happens to also reduce to a power law:

$$S(f) \simeq \frac{A}{B^{2q}} f^{p-2q}. \tag{15}$$

A transition between the two power laws occurs at intermediate frequencies ($Bf \sim 1$). To represent a spectrum such as (14) using Eqs. (7) and (8), it must be broken down into some finite number of bands, each of which can be approximated as a power law. Clearly, it is reasonable to represent frequencies $Bf << 1$ with one such band and $Bf >> 1$ with a second. The intermediate frequency region should be partitioned into a sufficient number of bands that the change in slope is well approximated. An example is shown in Fig. 3.

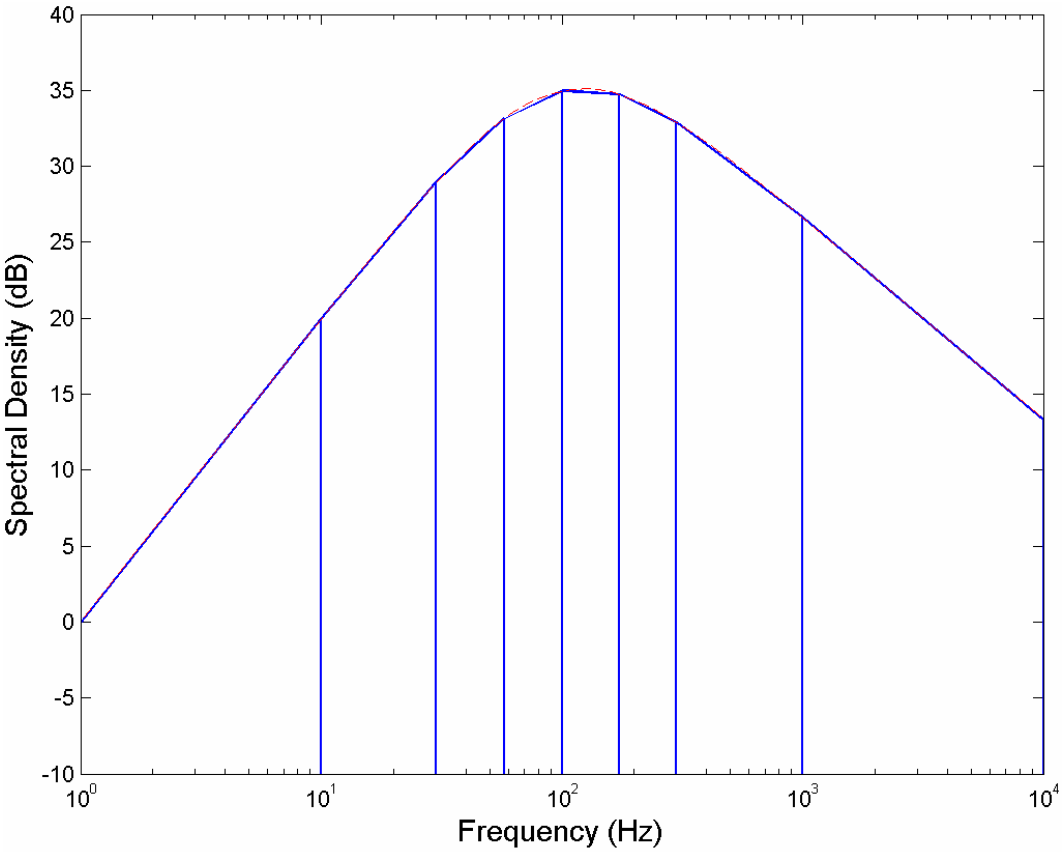

Figure 3. Illustration of approximating the dual power-law spectrum with a set of bands described by single power laws. This case has $A = 1$, $p = 2$, $b = 0.01$, and $\beta = 5/3$. The dual-power law is shown with the dashed line, and the bands for the approximation are solid lines.

c. Java Implementation

Having established that the power-law representation has sufficient flexibility to represent a wide variety of spectral models and frequency-dependent functions, we turn to the issue of implementation in software. This report describes such an implementation as a *class* in the Java programming language. A class represents the *variables* and *methods* (functions, procedures, or routines in the terminology of other programming languages) that are associated with an *object* (in this case the spectral representation).* The variables are structured in a manner similar to the

*Since this report describes a Java-language software implementation, it is rather difficult to avoid the jargon associated with that language and object-oriented programming in general. We have endeavored to define the terms as they occur, although some background in Java would certainly be helpful to understanding parts of the report.

one previously used in the Matlab-based ABFA software, but with some simplifications and improvements in flexibility. The following fields (variables) are stored:

- numBands: the number of frequency bands N.
- lowFreq: the lower frequency bounds $f_{\ell,n}$.
- highFreq: the upper frequency bounds $f_{u,n}$.
- specCoef: the coefficients A_n.
- specSlope: the power-law exponents p_n.
- phaz: the signal phases φ_n.
- dBref: the reference for conversions between decibels and spectral units.

The first of these is an integer; the middle five are $N \times 1$ double-precision floating-point arrays; the last is a double-precision floating-point value.

The phaz field was not present in the original ABFA representation. It has been added to accommodate cross spectra, coherent signals, and transfer functions. A field called OctaveBand, which contained information on the analysis bandwidth, has been removed, as that functionality will now be handled through the software interface. Also, a field called Active, which contained Boolean values representing whether the band was activated (turned on for calculations) has been removed. (Bands are now assumed to always be active.) The loudness field [which normally stored $E(f_{\ell,n}, f_{u,n})$ in dB] has been replaced by specCoef for simplicity.

In the remainder of this report, we refer to the Java class that defines these seven fields and the methods for manipulating them as the *BandedPowerLawSpec class*. The BandedPowerLawSpec class is used to create (instantiate) and manipulate *BandedPowerLawSpec objects*.

At the time of writing of this report, the BandedPowerLawSpec class is part of the package mil.army.usace.erdc.btraacoustics, which also includes

methods for reading signatures in HDF5 format, representing weighting curves for human audibility, and for calculating wind noise produced by atmospheric turbulence.

d. Constructors

In the Java programming language, a *constructor* creates and initializes new objects. Two BandedPowerLawSpec class constructors are available. One of these takes no argument and creates an object with no bands (numBands = 0). The second takes a single integer argument whose value is N, and creates an object with N bands. All five of the field arrays, lowFreq, highFreq, specCoef, specSlope, and phaz, are allocated and filled with zeros. Both constructor methods set dBref to its default value, 1.

Once a BandedPowerLawSpec object has been constructed, the fields must be set to their desired values. Shortly we will discuss a family of *low-level* methods that accomplish this task. However, it is envisioned that the approach of invoking the constructor followed by calls to the low-level methods will normally not be used directly. Instead, the BandedPowerLawSpec class provides many *high-level* methods for creating and intializing BandedPowerLawSpec objects. Most of the high-level methods accept the signal power over a convenient bandwidth to determine the power-law parameters. It is normally easier and less error prone to use these methods. They are described in more detail in Section 4.

e. Representing Frequency Bounds

The BandedPowerLawSpec class allows the frequency bounds $f_{\ell,n}$ and $f_{u,n}$ to be specified in three different formats. The first, and most obvious, format is direct specification as separate, N-element arrays.

The second format is a $2 \times N$ array. For example, suppose the variable `freqLims` is such a $2 \times N$ array. `freqLims[0]` would contain the lower frequency bounds and `freqLims[1]` the upper frequency bounds. The main utility of this format is that it allows a Java method, which is limited to producing a single output object, to return frequency bounds as its output.

The third format is a single $(N + 1)$-element array. The lower frequency bounds are the first N elements of this array; the upper frequency bounds

are the last N elements. Hence $f_{\ell,n+1} = f_{u,n}$. This format is more compact than the other two and again involves only a single object. However, it applies only to spectra that have contiguous bands (i.e., no gaps or overlaps between the bands).

f. Lower-Level Get/Set Methods

The lower-level set methods provide (essentially) direct specification of the lowFreq, highFreq, specCoef, specSlope, phaz, and dBref fields. The lower-level get methods retrieve the values of these fields.

The lower-level set methods are setFreqBounds, setSpecCoef, setSpecSlope, setPhase, and setDBref. The first of these accepts the frequency bounds in any of the three formats mentioned in the previous section. The next three accept the spectral coefficient, slope, and phase, respectively, as $N \times 1$ arrays. These methods all test whether the specified inputs are consistent with the size that was specified when the BandedPowerLawSpec was constructed. An error message is returned if not.

Note that the numBands field can only be set at the time of construction and cannot be changed by the user, since that would increase the likelihood of array indexing errors.

To illustrate the use of the constructor and low-level set methods, suppose we wish to construct a power-law spectrum that has $A_n = 6.2$ and $p_n = -5/3$ between the frequencies of 50 and 1000 Hz. Since only a single band is involved, we first construct the object with

```
BandedPowerLawSpec spec0 = new
                   BandedPowerLawSpec(1);
```

Next, we create double-precision Java arrays fl={50}, fu={1000}, sc={6.2}, and ss={−5/3}, which represent the lower frequency bounds, upper frequency bounds, spectral coefficients, and spectral slopes, respectively. For example,

```
double[] fl = {50};
```

Then, we call the set methods

```
spec0.setFreqBounds(fl, fu);

spec0.setSpecCoef(sc);

spec0.setSpecSlope(ss);
```

The desired BandedPowerLawSpec object has now been created.

As mentioned earlier, it is expected that the higher-level set methods will normally be used instead of the lower-level methods described here. The exceptions are setPhase and setDBref. The higher-level set methods do not alter the values of the phaz and dBref fields. If non-zero phases are desired, setPhase must be invoked. Similarly, if a decibel reference other than the default value of 1 is desired, setDBref must be invoked. (Conversions to and from decibels will be discussed later in this report.)

The lower-level get methods, which are self-explanatory, are getLowFreq, getHighFreq, getSpecCoef, getSpecSlope, getPhase, and getDBref.

g. Clones, Equality Tests, and String Conversions

The BandedPowerLawSpec class overrides the Java Object class methods clone, equals, and toString. It also provides additional functionality for comparing and copying BandedPowerLawSpec objects.

In Java, a *clone* refers to an exact copy of an object. (Simply setting one object equal to another creates two references to the same object, rather than a copy.) The BandedPowerLawSpec class provides its own clone method. It also provides a method called copy, which is the same as clone but recasts the Java object as a BandedPowerLawSpec object. Both clone and copy create a new object from the current instance. For example, if spec1 is a previously instantiated BandedPowerLawSpec object, the statement `spec2=spec1.copy()` makes a copy and places it in the BandedPowerLawSpec class object spec2. The method copyRev reverses this pattern by copying into the current instantiation: `spec1.copyRev(spec2)` makes a copy of spec2 and places it in spec1. (Both spec1 and spec2 must first be instantiated to invoke copyRev.)

The BandedPowerLawSpec class also provides a method called copyBand, which copies a single band from one BandedPowerLawSpec object to

another. The BandedPowerLawSpec objects must already be instantiated and have arrays allocated to hold the data. The user specifies the indices of the band in each object.

Three varieties of equality tests are provided by the BandedPowerLawSpec class. One is a conventional Java equals test for the BandedPowerLawSpec class, which tests whether all of the fields in two objects are identical. A second, called equalsBand, tests whether a particular band in one BandedPowerLawSpec object is equal to a particular band in another. The user specifies the indices of the bands to be tested. Lastly, a method called equalsFreqBounds is provided to test whether all of the frequency bounds (the lowFreq and highFreq fields) are identical between two objects.

The contents of a BandedPowerLawSpec object may be displayed with the toString method. For example,

```
System.out.println(spec1.toString());
```

displays the BandedPowerLawSpec object. Each band appears on a separate line. The fields are printed in the following order: lowFreq, highFreq, specCoef, specSlope, and phaz.

h. Evaluating the Spectrum

The method evalSpec takes as input an array of frequencies, with an arbitrary length M, and returns the values of the spectrum at those frequencies. The output is a $2 \times M$ array, where the first row represents the magnitude of the spectrum and the second its phase.

4 Signal Power and Moment Methods

This section describes methods for calculating and setting the signal power and moments of BandedPowerLawSpec objects. These are termed the *higher-level* get and set methods.

a. Higher-Level Set Methods

In many applications it is more convenient to specify the signal power in an analysis band than the coefficients A_n. The setSpecPower method provides this capability. It creates a new BandedPowerLawSpec object with the desired properties. In this sense, it replaces both the constructor and lower-level set functions.*

The main form of setSpecPower takes the following as input: (1) a scalar BW that describes the analysis bandwidth for the signal powers, (2) the frequency bounds $f_{\ell,n}$ and $f_{u,n}$, (3) an array specifying the signal powers ζ_n in each band, and (4) an array specifying the center frequencies $f_{c,n}$ in each band.

A form is also available that omits the explicit specification of the center frequencies, in which case the center frequencies are set equal to the average of the lower and upper bounds. The following explains how BW affects the mapping of ζ_n and $f_{c,n}$ into A_n and p_n:

- When BW = 0, an error is issued. BW is reset to 1 (unit bandwidth), and the ζ_n are interpreted as power in constant analysis bands, as described in the following.

- When BW > 0 but is finite in value, its value is interpreted as the power in a constant analysis bandwidth Δf such that $\zeta_n = E(f_{\ell,n}, f_{\ell,n} + \Delta f)$. The input $f_{c,n}$ have no effect and the p_n are all set to zero. In this case, $E(f_{\ell,n},$

*Note that setSpecPower and the other higher-level set methods described in this section extend the functionality normally provided by a Java set method, in that they also construct the Spec object. This extension is simply a matter of convenience; it is not anticipated that users will have a need to separately construct the objects and set their properties.

$f_{\ell,n} + \Delta f) = A_n \Delta f$ is independent of frequency. Hence, the ζ_n contain the power in a fixed analysis bandwidth Δf.

- When BW < 0, its value is used to calculate an analysis bandwidth for proportional bands according to the formula $\Delta f = (2^{|BW|/2} - 2^{-|BW|/2})f$. Thus, a value BW = –1 represents octave bands, whereas BW = –1/3 represents 1/3-octave bands. The values ζ_n are interpreted as $E\,(2^{-|BW|/2}f,\ 2^{|BW|/2}f)$ and the p_n are all set to –1. In this case, $E\,(2^{-|BW|/2}f,\ 2^{|BW|/2}f) = A_n|BW|\log 2$ is independent of frequency. Hence ζ_n describes the power per octave (or other fractional octave) band. The values of the $f_{c,n}$ have no effect.

- When BW = ∞, $\zeta_n = E\,(f_{\ell,n}, f_{u,n})$, the total energy in the signal band. The values of A_n and c are determined from the ζ_n and the $f_{c,n}$.

Let us consider some examples. Suppose we wish to construct a white noise spectrum between 0 and 100 Hz, with a power of 3.0 relative to 1-Hz bands. This can be accomplished with the following command, where fl={0.0}, fu={100.0}, and powr={3.0}:

```
BandedPowerLawSpec spec1=setSpecPower(1.0,fl,fu,
                   powr);
```

Here `spec1` is the new BandedPowerLawSpec object. Pink noise, extending from 10 to 1000 Hz, and with a power of 3.0 per 1/3-octave band, can be created with

```
BandedPowerLawSpec spec2=setSpecPower(-1.0/3.0,
                   fl,fu,powr);
```

where spec2 is the new BandedPowerLawSpec object, fl={10.0}, fu={1000.0}, and powr={3.0}. A spectrum with three narrow (0.2-Hz) lines, centered at 20, 40, and 60 Hz, and having a power of 1.0, 3.0, and 2.0, respectively, can be created as follows:

```
BandedPowerLawSpec spec3=setSpecPower(Double.
                   POSITIVE_INFINITY,fl,fu,powr);
```

where spec3 is the new BandedPowerLawSpec object, fl={19.9,39.9,59.9}, fu={20.1,40.1,60.1}, and powr={1.0,3.0,2.0}. Replacing the first argument (BW) by 0.2 would have accomplished the same purpose. Lastly, consider the creation of octave band data with geometric center frequencies fg={20.0,40.0,80.0} and power in the bands of powr={1.0,3.0,2.0}. The lower frequency bounds fl are set to fg divided by the square root of 2, and the upper frequency bounds fu are fg times the square root of 2. The command

```
BandedPowerLawSpec spec4=setSpecPower(Double.
                   POSITIVE_INFINITY,fl,fu,powr);
```

creates the desired object with slopes $p_n = 0$. Alternatively,

```
BandedPowerLawSpec spec4=setSpecPower(-1.0,fl,fu,
                   powr);
```

could be used to create bands with slopes $p_n = -1$. BW = ∞ and BW = –1 give the same result in this case, since the frequency bands are octave bands.

The higher-level set methods setSpecCoefAndSlope and setSpecEndPoints provide alternatives to specifying signal power with the setSpecPower method. The method setSpecCoefAndSlope directly takes input arrays specifying the lower frequency bounds, upper frequency bounds, spectral coefficients, and (optionally) spectral slopes. Its main utility is for creating power-law spectral models. For example, for the situation described in Section 3f, once the arrays fl, fu, sc, and ss have been created, we could set up the object in one step as

```
BandedPowerLawSpec spec0=setSpecCoefAndSlope(fl,fu,
                   sc,ss);
```

The method setSpecEndPoints takes as input an $(N + 1) \times 1$ array of contiguous bounding frequencies and an $(N + 1) \times 1$ array of powers at these frequencies. The values of A_n and p_n are determined by matching the power laws in each band at the end points. On a log-log plot, this can be interpreted as simply drawing linear segments connecting the specified frequencies and powers.

Numerous other higher-level set methods are available that create BandedPowerLawSpec objects by invoking setSpecPower, setSpecCoefAndSlope, or setSpecEndPoints. They are associated with many of the common spectral models described in Section 3b. White noise, pink noise, fractional octave bands, standard octave bands, standard one-third octave bands, evenly spaced bands, and dual-power law spectra can be created directly. For example, the following command produces the white noise spectrum described earlier and is somewhat more convenient than calling set setSpecPower:

```
BandedPowerLawSpec spec1=setWhiteNoise
                   (0.0,100.0,3.0);
```

More information on these short-cut set methods is provided in the Java class documentation.

b. Higher-Level Get Methods

The method getSpecPower complements setSpecPower. Like setSpecPower, it takes as input a scalar BW that controls the interpretation of the signal powers. In its basic form, though, there are no other input arguments; it operates on the current BandedPowerLawSpec object instance. The output of getSpecPower is an N-length array of signal powers ζ_n for each band in the current instance. The interpretation of BW is as follows:

- When BW = 0, $\zeta_n = 0$ is returned.
- When BW > 0 but is finite in value, the p_n are all assumed to equal 0. The value of BW is interpreted as a constant analysis bandwidth Δf, and $\zeta_n = E\,(f_{\ell,n}, f_{\ell,n} + \Delta f)$ is returned.
- When BW < 0, the p_n are all assumed to equal –1. The value of BW is used to calculate an analysis bandwidth for proportional bands according to the formula $\Delta f = (2^{|\mathrm{BW}|/2} - 2^{-|\mathrm{BW}|/2})f$, and $\zeta_n = E\,(2^{-|\mathrm{BW}|/2}f, 2^{|\mathrm{BW}|/2}f)$ is returned.
- When BW = ∞, $\zeta_n = E\,(f_{\ell,n}, f_{u,n})$ (the total power in the signal band) is returned. This value depends on the A_n and p_n for the bands.

Another form of the getSpecPower method omits the specification of BW and simply returns the total power in each band (as corresponds to BW = ∞). The methods getSpecMoment (which takes an integer m as an argument, indicating the desired moment) and getCenterFreq similarly return the spectral moments and center frequencies corresponding to each band. For example,

```
spec1.getSpecPower(1)
```

returns the power per unit bandwidth in each band of the BandedPowerLawSpec object spec1 (assuming the p_n are zero),

```
spec1.getSpecPower()
```

or

```
spec1.getSpecPower(Double.POSITIVE_INFINITY)
```

returns the total power in each band, and

```
spec1.getCenterFreq()
```

returns the center frequency for each band.

The center frequency $f_{c,n}$ of a band is defined by the BandedPowerLawSpec class as $M_n^1\left(f_{\ell,n}, f_{u,n}\right) / M_n^0\left(f_{\ell,n}, f_{u,n}\right)$. Note that when $p_n = 0$, from (5) we have

$$f_{c,n} = \frac{1}{2}\frac{f_{u,n}^2 - f_{\ell,n}^2}{f_{u,n} - f_{\ell,n}} = \frac{1}{2}\left(f_{u,n} + f_{\ell,n}\right). \tag{16}$$

That is, the center frequency is the average of the bounds. The center frequency as defined here is not normally equal to the geometric center frequency mentioned in Section 2 in connection with proportional band data, namely $\sqrt{f_{\ell,n} f_{u,n}}$.

Finally, there are forms of getSpecPower, getSpecMoment, and getCenterFreq that take as input an arbitrary lower frequency f_ℓ and upper frequency f_u. These are useful if one wants to calculate, for example, the total

signal power between two frequencies, regardless of the positions of the bands in the representation. Calculation of moments for such a general situation where the frequencies do not align with the bands of the representation is rather complicated. The frequencies may be positioned within the bands and may even span multiple bands. In general,

$$\begin{aligned} M^m(f_\ell, f_u) &= \int_{f_\ell}^{f_u} \sum_{n=1}^{N} f^m S_n(f)\, df \\ &= \sum_{n=1}^{N} \int_{f_\ell}^{f_u} f^m S_n(f)\, df. \end{aligned} \tag{17}$$

Defining

$$\overline{f}_{\ell,n} = \max\left(f_\ell, f_{\ell,n}\right) \tag{18}$$

and

$$\overline{f}_{u,n} = \min\left(f_u, f_{u,n}\right), \tag{19}$$

one can show that

$$\int_{f_\ell}^{f_u} f^m S_n(f)\, df = \begin{cases} \int_{\overline{f}_{\ell,n}}^{\overline{f}_{u,n}} A_n f^{p_n+m}\, df, & \overline{f}_{\ell,n} < \overline{f}_{u,n} \\ 0, & \overline{f}_{\ell,n} \geq \overline{f}_{u,n} \end{cases}. \tag{20}$$

This leads to the following recipe for calculating the moments between two arbitrary frequencies:

1. Change the lowFreq field to the values given by (18). Change the highFreq field to the values given by (19). If the value in the highFreq field is less than the value in lowFreq, set the values in the two fields so they are equal (zero bandwidth).

2. Add *m* to each element in the specSlope field.

3. Call getSpecPower to determine the power in each band.

4. Return the sum of the powers returned by getSpecPower.

c. Decibel Conversions

As mentioned earlier, spectra are often represented with decibels. The methods dBConvTo and dBConvFrom convert between signal power in decibels and in amplitude squared units. In their basic, static form, each of these take two inputs: an array (or scalar) containing the powers, followed by the decibel reference value (in squared amplitude units) to be used for the conversion. They then perform the conversion to or from decibels and return the result as an array (or scalar, if that was the input). If a reference value of 0 is specified, no conversion is performed. If a negative value is specified, the absolute value is used.

Alternatively, dBConvTo or dBConvFrom may be invoked as instance methods. Then the reference value is not explicitly provided, and the implicit value for the BandedPowerLawSpec object is used. The implicit value can be set by the user with the setDBref method.

For convenience, the BandedPowerLawSpec class defines a number of static variables containing commonly used reference values. As an example, suppose a BandedPowerLawSpec object spec1 has been created and one wishes to associate with it the normal decibel reference for water acoustics, 1 μPa. This would be done as show below:

```
spec1.setDBref(Math.pow(dBrefWater,2.0));
```

Then, any time the instance form of dBConvTo or dBConvFrom is called to convert to or from decibels, the reference value for water acoustics would be used, unless it is explicitly overridden.

Consider next the example in Section 4a, where we created an octave-band BandedPowerLawSpec object. If we wished to create an object with the same frequency bands but with powers of 10, 30, and 20 dB relative to the normal dB reference for air acoustics (20 μPa), we could use the command

```
BandedPowerLawSpec spec4=setSpecPower(-1.0,fl,fu,
                   dBConvFrom(powrDB,Math.pow(dBrefA
                   ir,2.0)));
```

where powrDB={10.0,30.0,20.0}.

5 Filtering and Frequency Bands

Up to this point, we have described how the BandedPowerLawSpec class represents spectra, how BandedPowerLawSpec objects are created, and how their properties are set and retrieved. The main utility of the BandedPowerLawSpec class lies, however, in manipulating the spectra in various ways. This section describes operations that involve a single spectrum: filtering its frequency content and converting it to a different set of frequency bands. All of these are instance methods, meaning that they operate on the current object instance. The next section will describe operations involving multiple objects.

a. Removing Bands and Eliminating Overlap

The filterCull method removes bands from a BandedPowerLawSpec object whose power is below a specified threshold. If no threshold is passed to the method, the threshold is assumed to be zero, in which case bands having no power (equivalently, $A_n = 0$) are removed.

The methods for creating BandedPowerLawSpec objects generally permit overlapping bands to be created. The overlap can be removed by invoking the method filterRemOverlap. In regions where there is overlap, this method will create a single, new band by summing the powers from the overlapping bands. The frequency bounds of the overlapping bands are changed so there is no longer overlap.

b. Low-pass, High-pass, Bandpass, and Stopband Filtering

Filtering involves removal of signal power over a range of frequencies. A low-pass filter retains signal power below a specified frequency. A high-pass filter retains signal power above a specified frequency. A bandpass filter retains signal power between a pair of specified frequencies. A stopband filter removes all signal energy between a pair of frequencies. The filterLowPass, filterHighPass, filterBandPass, and filterBandStop methods implement these operations.

The low-pass filter can be considered a special case of the bandpass filter with the passband starting at 0. Similarly, the high-pass filter is a band-

pass filter with the passband ending at infinite frequency. The stopband filter can also be considered a passband filter, where the frequency ranges outside of the stopband become the passbands. Since all of the other filters can thus be considered special cases of the bandpass filter, we describe here only the implementation of the bandpass filter. In the following discussion, the user-specified passband is $[f_\ell, f_u]$:

- Find the last band m for which $f_\ell \geq f_{\ell,m}$. Remove all bands preceding this one. If $f_\ell \geq f_{u,m}$ (indicating that f_ℓ falls in a gap between bands), remove this band also. Otherwise, change the lower frequency of this band to f_ℓ.

- Find the first band n for which $f_u \leq f_{u,n}$. Remove all bands following this one. If $f_u \leq f_{\ell,n}$ (indicating that f_u falls in a gap between bands), remove this band also. Otherwise, change the upper frequency of this band to f_u.

For the filterBandPass and filterBandStop methods, either single values or arrays may be input for f_ℓ and f_u. Arrays are used to create multiple passbands or stopbands. Only single values may be passed to filterLowPass and filterHighPass. As an example, the code

```
spec1.filterLowPass(100.0);
```

removes all signal power above 100.0 Hz from the BandedPowerLawSpec object spec1. The code

```
spec1.filterBandPass(fl, fu);
```

where fl is an array containing {20,60}, and fu is an array containing {40,80}, removes all signal power except that between 20 and 40 Hz and 60 and 80 Hz.

The filtering procedure described here is perfect in the sense that all power outside of the passband is removed without affecting the power within the passband. Actual filters do not achieve this perfect characteristic. To implement an actual filter, one can multiply two BandedPowerLawSpec objects, as will be described later.

c. Converting Frequency Bands

Suppose one wishes to change the frequency bounds (the lowFreq field, containing the $f_{\ell,n}$, and the highFreq field, containing the $f_{u,n}$) of a spectral representation. For example, one may wish to convert from frequency bands with a constant analysis bandwidth to proportional bands. Without loss of generality, we can consider conversion of a spectral representation to a single frequency band $[f_\ell, f_u]$; the case of converting to a multiple-banded representation with new frequencies would just be a repetition of the process for a single band.

If the frequencies f_ℓ and f_u are contained within a single band of the initial representation, the conversion is straightforward: we simply copy the values of A_n and p_n and replace the former values of $f_{\ell,n}$ and $f_{u,n}$ with f_ℓ and f_u.

When $[f_\ell, f_u]$ spans multiple frequency bands, conversion becomes more complicated. It is not possible to equate a single power-law band to two or more power-law bands, unless the two bands happen to have the same slope p_n. In general, the conversion can only be performed approximately. The procedure we have adopted is based on preserving the zeroth moment (energy) and first moment of the representation.* To do this, one first calculates $M^0(f_\ell, f_u)$ and $M^1(f_\ell, f_u)$ using the methods described in Section 4. These values can then be used to determine A and p in the new representation. From (5),

$$M^0(f_\ell, f_u) = \begin{cases} \frac{A}{p+1}\left(f_u^{p+1} - f_\ell^{p+1}\right), & p \neq -1 \\ A\log(f_u / f_\ell), & p = -1 \end{cases} \tag{21}$$

and

$$M^1(f_\ell, f_u) = \begin{cases} \frac{A}{p+2}\left(f_u^{p+2} - f_\ell^{p+2}\right), & p \neq -2 \\ A\log(f_u / f_\ell), & p = -2 \end{cases}. \tag{22}$$

*The band conversion method described here, based on equating moments, is one possibility of many. An example alternative is minimization of the squared difference, integrated over frequency, between the two representations. The method described here was chosen for its simplicity and generality.

Taking the ratio $M^1(f_\ell, f_u) \,/\, M^0(f_\ell, f_u)$, we need to solve the following equation for p:

$$\frac{M^1(f_\ell, f_u)}{M^0(f_\ell, f_u)} = \begin{cases} \frac{p+1}{p+2}\frac{f_u^{p+2} - f_\ell^{p+2}}{f_u^{p+1} - f_\ell^{p+1}}, & p \neq -1 \text{ or } -2 \\ \frac{1}{p+2}\frac{f_u^{p+2} - f_\ell^{p+2}}{\log(f_u / f_\ell)}, & p = -1 \\ (p+1)\frac{\log(f_u / f_\ell)}{f_u^{p+1} - f_\ell^{p+1}}, & p = -2 \end{cases} . \tag{23}$$

This equation must generally be solved numerically. Once p has been determined, we can calculate A by solving for it in (21):

$$A = \begin{cases} (p+1) M^0(f_\ell, f_u) / \left(f_u^{p+1} - f_\ell^{p+1} \right), & p \neq -1 \\ M^0(f_\ell, f_u) / \log(f_u / f_\ell), & p = -1 \end{cases} . \tag{24}$$

The method convFreqBands implements the operation just described. This method can be invoked with any of the three formats described in Section 3e for representing the target frequency bounds.

6 Multi-Spectra Operations

This section describes operations involving two BandedPowerLawSpec objects. The main operations are incoherent addition of spectra and coherent multiplication and division. Each of these three operations are implemented in the BandedPowerLawSpec class with a static method and an instance method. An example of the static form is

```
spec3 = addSpectraInc(spec1, spec2);
```

where spec1 and spec2 are existing BandedPowerLawSpec objects, and spec3, their incoherent addition, is created. An example of the instance form is

```
spec1.addSpectraInc(spec2);
```

where spec1 and spec2 are again existing BandedPowerLawSpec objects, and the object spec1 is changed by incoherently adding spec2 to it.

The multi-spectra operations require that the frequency bounds, $f_{\ell,n}$ and $f_{u,n}$, are the same for both BandedPowerLawSpec objects. If the two objects do not initially meet this requirement, they are first converted to a common set of frequency bounds. We discuss this topic before describing the available multi-spectra operations.

a. Developing Common Frequency Bounds

This section describes how a set of common, bounding frequencies are devised for a pair of BandedPowerLawSpec objects. The procedure is illustrated in Figure 4.

An array of frequencies f_b is created containing all the unique values of $f_{\ell,n}$ and $f_{u,n}$ from both spectral representations. The array is sorted in order of increasing frequency, and repetitions are removed. The lower frequency bounds for a common representation can now be defined as $f_{\ell,n} = f_{b,n}$, $n = 0, \ldots, N-1$, where $N+1$ is the total number of bounding frequencies. The upper frequency bounds are $f_{u,n} = f_{b,n}$, $n = 1, \ldots, N$.

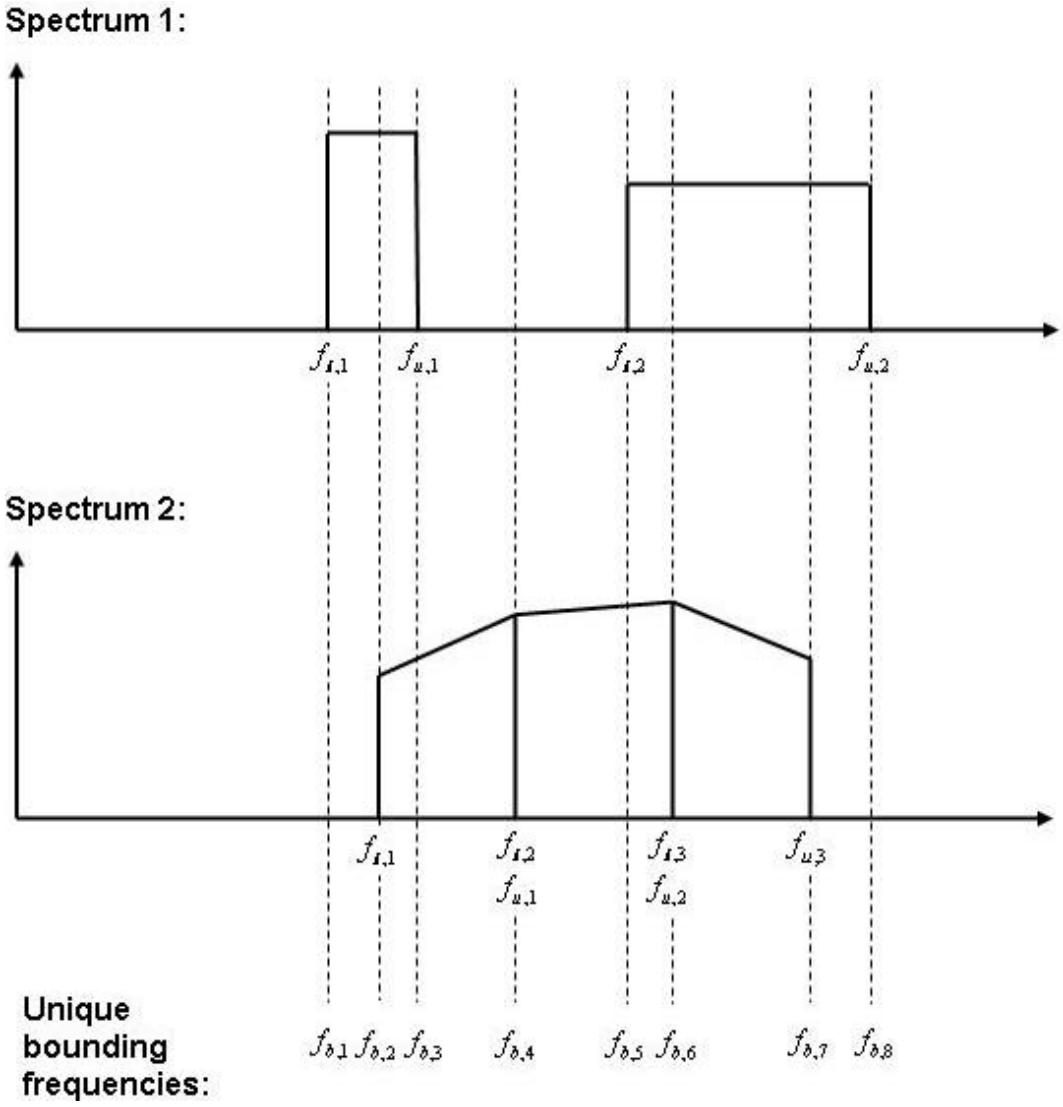

Figure 4. Determination of the unique bounding frequencies from a pair of Spec objects.

Once the common bounding frequencies have been determined, it is straightforward to individually convert the two BandedPowerLawSpec objects to these new bands. Note that each frequency band in the original spectral representations contains one or more frequency bands, or sub-bands, based on the new frequency bounds. Each of these subbands inherits the values of A_n and p_n from its parent. Any new frequency bands outside of those in the parent representation take $A_n = 0$.

b. Adding Spectra Incoherently

Suppose two autospectra that are to be added have the banded power-law form of (7) and (8). For example, we might want to add the autospectra of two types of uncorrelated noise to find the overall noise background. Such operations are not readily supported by the power-law form. If the power-law exponents in the two spectra being added differ, their summation can no longer be described by a plain power law. Therefore, summations involving differing exponents must be handled approximately.

The process of adding two spectra begins by finding the common frequency bands and partitioning the spectra into these new bands, as described in Section 6a. Once this has been done, common bands from the two spectra are independently added. In the following discussion, we assume that the spectra have been processed into such common bands, and therefore we need consider only the summation of one band with another that shares the same bounding frequencies f_ℓ and f_u. This process is repeated for each of the bands. Let us designate the spectral coefficients for the two spectra as A_a and A_b. The spectral slopes are p_a and p_b. Therefore, the summation is

$$S(f) = A_a f^{p_a} + A_b f^{p_b}. \tag{25}$$

Since the summation cannot be represented directly in the form $S(f) = Af^p$ (unless $p_a = p_b$), we have chosen to assign A and p in a way that preserves the zeroth and first moments of the summation. Integrating both sides of (25) from f_ℓ to f_u, we have

$$M^0(f_\ell, f_u) = M_a^0(f_\ell, f_u) + M_b^0(f_\ell, f_u). \tag{26}$$

Multiplying both sides of (25) by f and again integrating from f_ℓ to f_u, we have

$$M^1(f_\ell, f_u) = M_a^1(f_\ell, f_u) + M_b^1(f_\ell, f_u). \tag{27}$$

Hence, to determine p in the approximate spectral summation of the form $S(f) = Af^p$, we can apply (23) with

$$\frac{M^1(f_\ell, f_u)}{M^0(f_\ell, f_u)} = \frac{M_a^1(f_\ell, f_u) + M_b^1(f_\ell, f_u)}{M_a^0(f_\ell, f_u) + M_b^0(f_\ell, f_u)}. \tag{28}$$

Next, A follows from (24).

c. Dividing Spectra

Often it is desirable to take the ratio of one spectrum or frequency-dependent function to another. This happens, for example, when we calculate the signal-to-noise ratio, which plays a key role in signal detection. Another example is the calculation of a transfer function by dividing the cross spectrum between two signals by an autospectrum.

As with the addition of spectral bands, we first repartition the two spectra into common frequency bands. Then, on a band-by-band basis, the ratio can be calculated simply as $A_a f^{p_a} / A_b f^{p_b}$, which is equivalent to setting $A = A_a/A_b$ and $p = p_a - p_b$. The overall phase angle is $\varphi = \varphi_a - \varphi_b$. The BandedPowerLawSpec class includes methods that implement division in this manner. The method divSpectra is the static version, divSpectraNum is the instance version with the current instance in the numerator, and divSpectraDen is the instance version with the current instance in the denominator.

d. Multiplying Spectra

Multiplication of spectra or frequency-dependent functions is also a common operation. It may occur, for example, when we wish to multiply an autospectrum by a transfer function representing a filter. As with the division of spectra, this operation is straightforward if we first repartition the two spectra into common frequency bands. Then, on a band-by-band basis, the product is $\left(A_a f^{p_a}\right)\left(A_b f^{p_b}\right)$, which is equivalent to setting $A = A_aA_b$ and $p = p_a + p_b$. The phase angle is $\varphi = \varphi_a + \varphi_b$.

7 Conclusion

This report has described the software design and underlying mathematics of an object-oriented approach to creating and manipulating spectra. The new software provides a high degree of flexibility for representing common spectral models such as evenly spaced bands, octave bands, narrow spectral lines, broadband noise, and power laws. A library of methods has been developed to easily set up such spectral models. Conversions between the various spectral models are also easily performed. Many operations on spectra, such as filtering, incoherent addition, and application of transfer functions, can be applied.

This capability will serve as a foundation for future development of acoustic and seismic tactical decision aids and mission planning tools, such as ERDC Battlefield Terrain Reasoning and Awareness and its successors. Extensions to the class have already been written to represent audibility weighting curves and wind noise produced by atmospheric turbulence. Another potential acoustical application is the modeling of outdoor noise propagation and annoyance. Because of its generality, the new software could potentially be used for many other problems involving spectral manipulations.

References

Bendat, J.S., and A.G. Piersol. 1986. *Random Data: Analysis and Measurement Procedures.* New York: Wiley-Interscience.

von Kármán, T. 1948. Progress in the statistical theory of turbulence. *Journal of Marine Research,* 7: 252–264.

Wilson, D.K. 2006. Sensor Performance Evaluator for Battlefield Environments (SPEBE) tutorial. ERDC/CRREL TR-06-12. Hanover, NH: U.S. Army Engineer Research and Development Center, Cold Regions Research and Engineering Laboratory.

Wilson, D.K., V.A. Nguyen, N. Srour, and J. Noble. 2002. Sound exposure calculations for transient events and other improvements to an acoustical tactical decision aid. ARL-TR-2757. Adelphi, MD: U.S. Army Research Laboratory.

Wilson, D.K., and G.L. Szeto. 2000. Reference guide for the acoustic battlefield aid (ABFA), version2. ARL-TR-2159. Adelphi, MD: U.S. Army Research Laboratory.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
December 2006	Technical Report	

4. TITLE AND SUBTITLE

Object-Oriented Approach to Manipulating Acoustic and Seismic Spectra

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

D. Keith Wilson and Jacob I. Torrey

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U.S. Army Engineer Research and Development Center
Cold Regions Research and Engineering Laboratory
72 Lyme Road
Hanover, NH 03755-1290

8. PERFORMING ORGANIZATION REPORT NUMBER

ERDC/CRREL TR-06-20

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of the Chief of Engineers
Washington, DC

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

Available from NTIS, Springfield, Virginia 22161.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The software design and underlying mathematics for an object-oriented, Java-based approach to creating and manipulating frequency-dependent functions, such as power spectral densities, is described. The frequency dependence is modeled as a series of power-law bands, which provides a high degree of flexibility and efficiency for representing common spectral models such as evenly spaced bands, octave bands, narrow spectral lines, broadband noise, and power laws. Conversions between the various spectral models are easily performed. Many common operations on spectra, such as filtering, incoherent addition, application of transfer functions, and calculation of signal-to-noise ratios, can be conveniently applied. While this capability was developed to serve as a basis for future development of tactical decision aids and mission planning tools for battlefield seismics and acoustics, many other applications involving spectra are possible.

15. SUBJECT TERMS

Acoustic spectra
Object-oriented modeling
Seismic spectra
Spectral models

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(include area code)*
U	U	U	U	43	